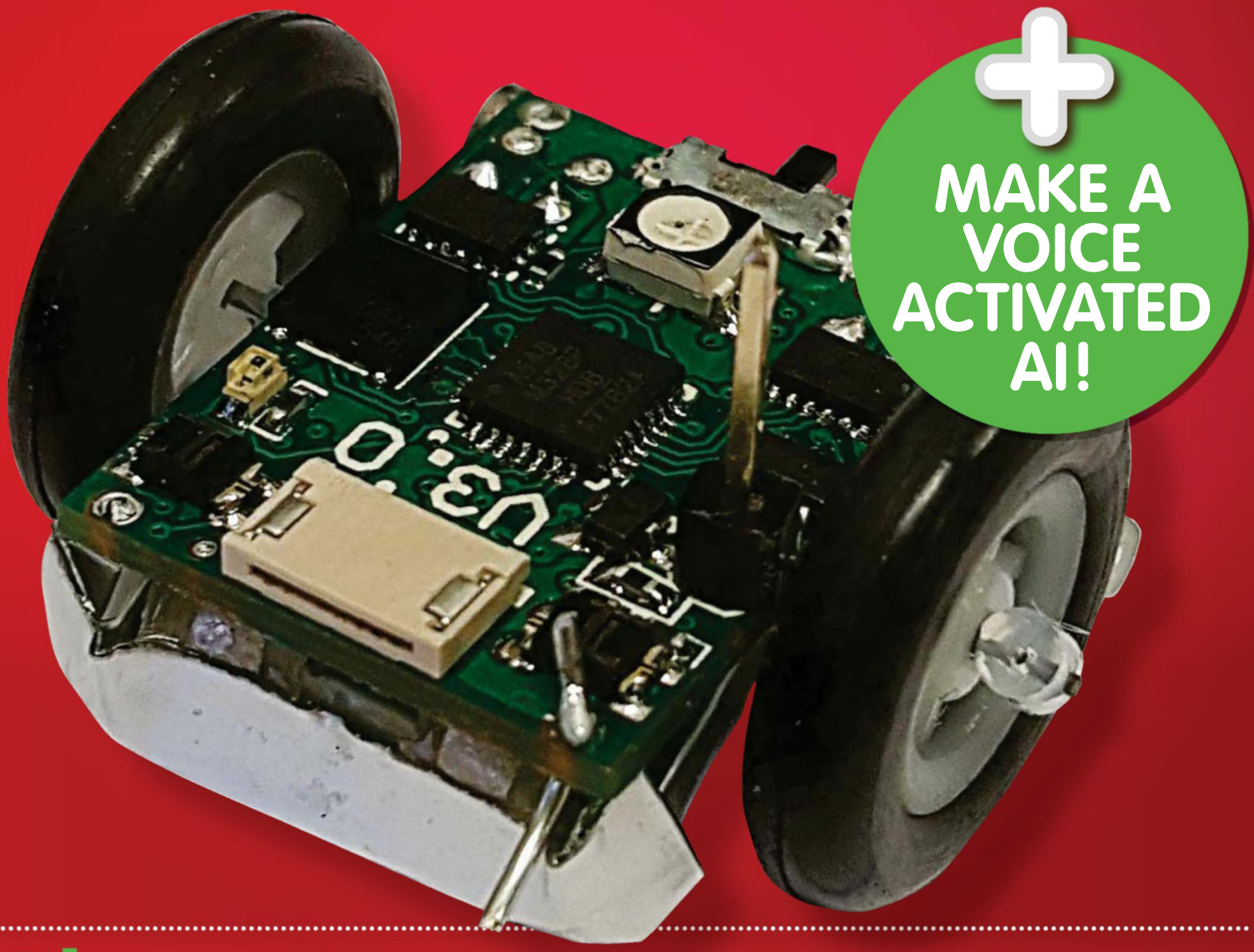
RasPi
DESIGN
BUILD
CODE
42
Get hands-on with your Raspberry Pi

ATTACK OF THE PI MICROBOTS



MAKE A VOICE ACTIVATED AI!

Plus Build pyramids in Minecraft

Welcome

The Raspberry Pi's potential for innovation in the field of education is huge. We thought we'd kick off our first issue of the new year by taking a look at one particularly innovative advance pioneered by the Imperial Robotics Society from Imperial College, London. Swipe towards the left a few times and find out how Joshua Eldon utilised Pi-powered micro-robots to teach robotic behaviour classes providing multiple systems for testing while using less space for the arenas.

If that's not enough ingenuity for you we have some more that you can do yourself, such as adding a voice-activated AI to your Google Assistant and building pyramids in Minecraft.

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

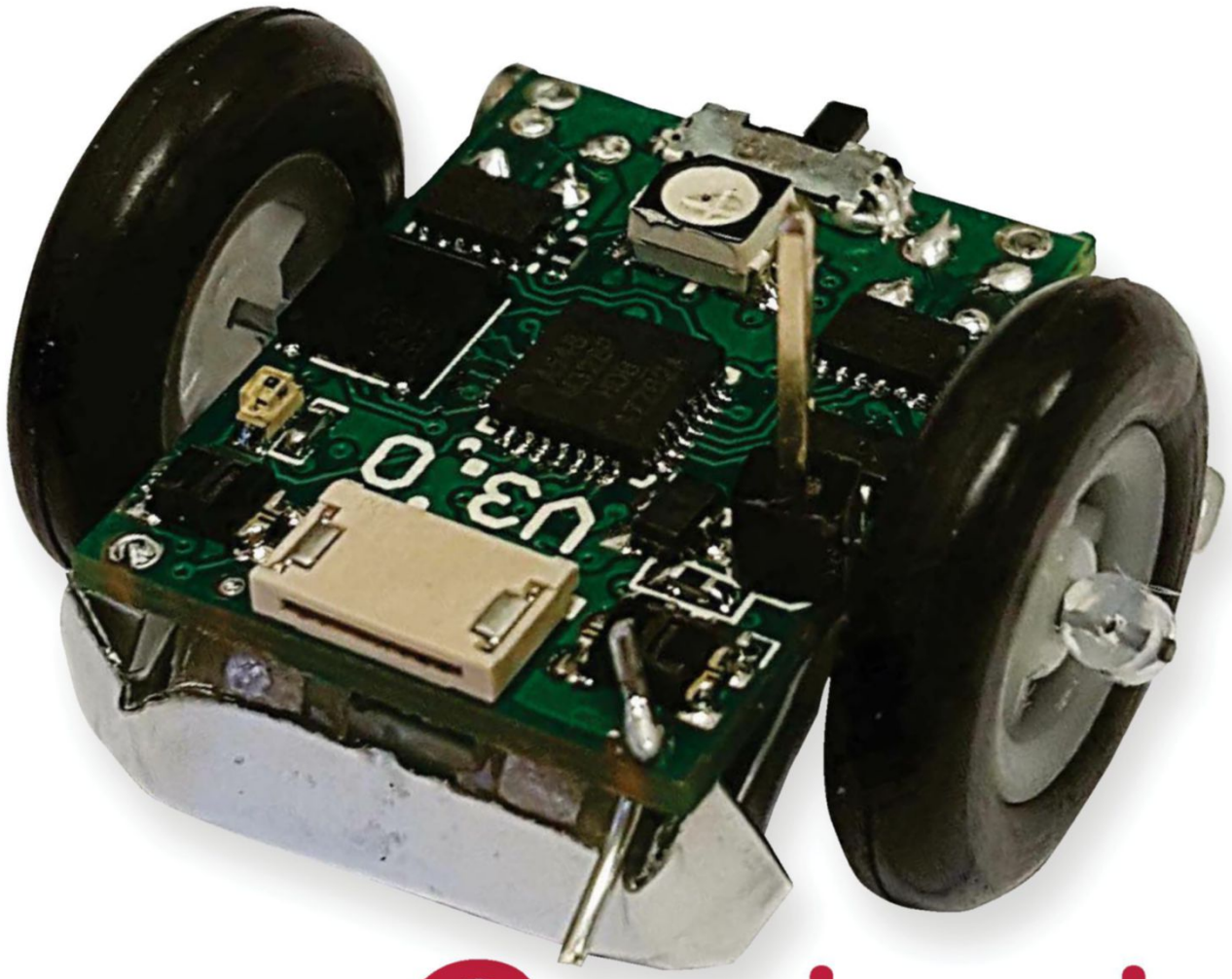
Editor

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com



Contents

Create 3D art pieces in Minecraft using Python loops

Taking our Minecraft pixel art into the third dimension, this time we'll build pyramids with Python loops

In previous issues, we started our series on Minecraft pixel art, by coding some 2D art blocks. This time we're taking a different approach to our art, by implementing an additional axis and therefore bringing it into the third dimension. Well, technically, everything is 3D in Minecraft, but previously we built some rather convincing 'flat' pixel art. Now we're taking it to the next level.

We're going to build a pyramid using while and for loops in Python. This will save us typing lots of similar lines of code. Spawning our coded creations is so much faster than placing each individual block manually in Minecraft's Creative mode.

If you're using Minecraft Pi edition on a Raspberry Pi, no additional software is necessary. We've also put together a number of tools to ensure this hack works on Linux, with a retail version of Minecraft. Therefore as a prerequisite, we assume you've installed McPiFoMo from our previous three issues. McPiFoMo includes MCPiPy by 'fleap' and 'bluepillRabbit' of MCPiPy.com; and Raspberry Jam, developed by Alexander Pruss.

All Python scripts should be saved in the directory

THE PROJECT ESSENTIALS

Raspberry Pi

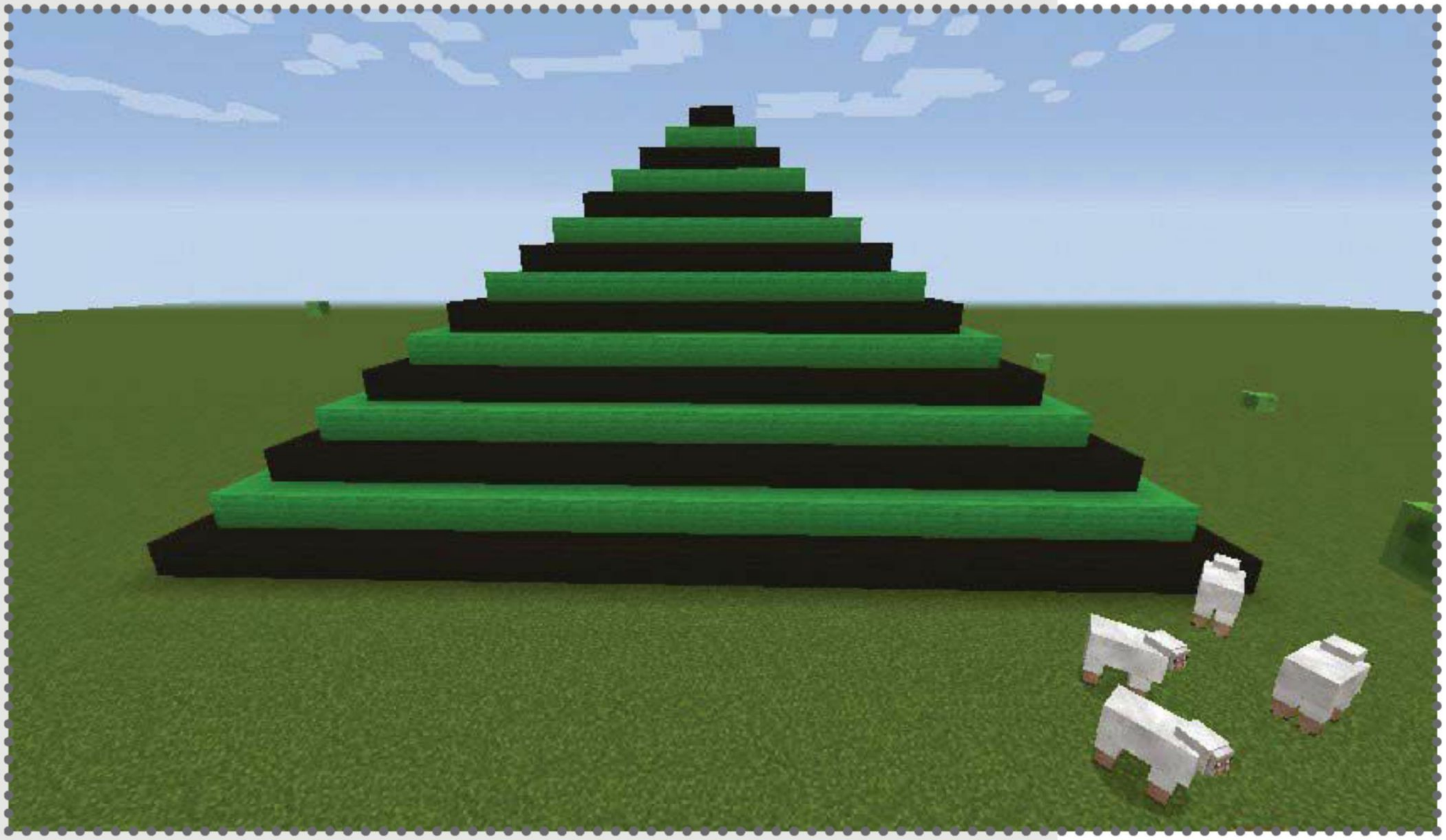
Enviro pHAT
http://bit.ly/EnviropHAT

Minecraft
www.mojang.com/games

Python
www.python.org

McPiFoMo
http://rogerthat.co.uk/McPiFoMo.rar

Block IDs:
http://bit.ly/MinecraftIDList



~/home/.minecraft/mcpipy/, regardless of whether you're running Minecraft Pi edition or retail Linux Minecraft. Be sure to run Minecraft with the Forge 1.8 profile that's included in our McPiFoMo package.

01 Starter code and variables

Here's the first block of code to start with.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
pos = mc.player.getTilePos()
x = pos.x + 2
y = pos.y
z = pos.z
height = 29
count = 0
blockID = 24
blockType = 1
```

02 Starter pyramid

First, we're going to initiate a bunch of variables, collecting the player's standing position with pos = mc.player.getTilePos(), and breaking that down into x, y, z coordinates with x = pos.x + 2, y = pos.y and z = pos.z. We've also got variables for the height of our pyramid; we'll get to shortly. Below that we have blockID and blockType, which will affect the blocks used to create our pyramid. Block ID 24:1 would be broken down to blockID 24, blockType 1. We felt Chiseled Sandstone looked quite 'pyramid-y' to begin with.

03 Adding the loops

Let's create some for loops nested within a while loop.

```
  while height - (2 * count) > 0:
    for block in range(height - (2 * count)):
        for row in range(height - (2 * count)):
            blockX = x + block + count
            blockY = y + count
            blockZ = z + row + count
            mc.setBlock(blockX,  blockY,  blockZ,
blockID, blockType)
    count += 1
```

04 Customising our pyramid

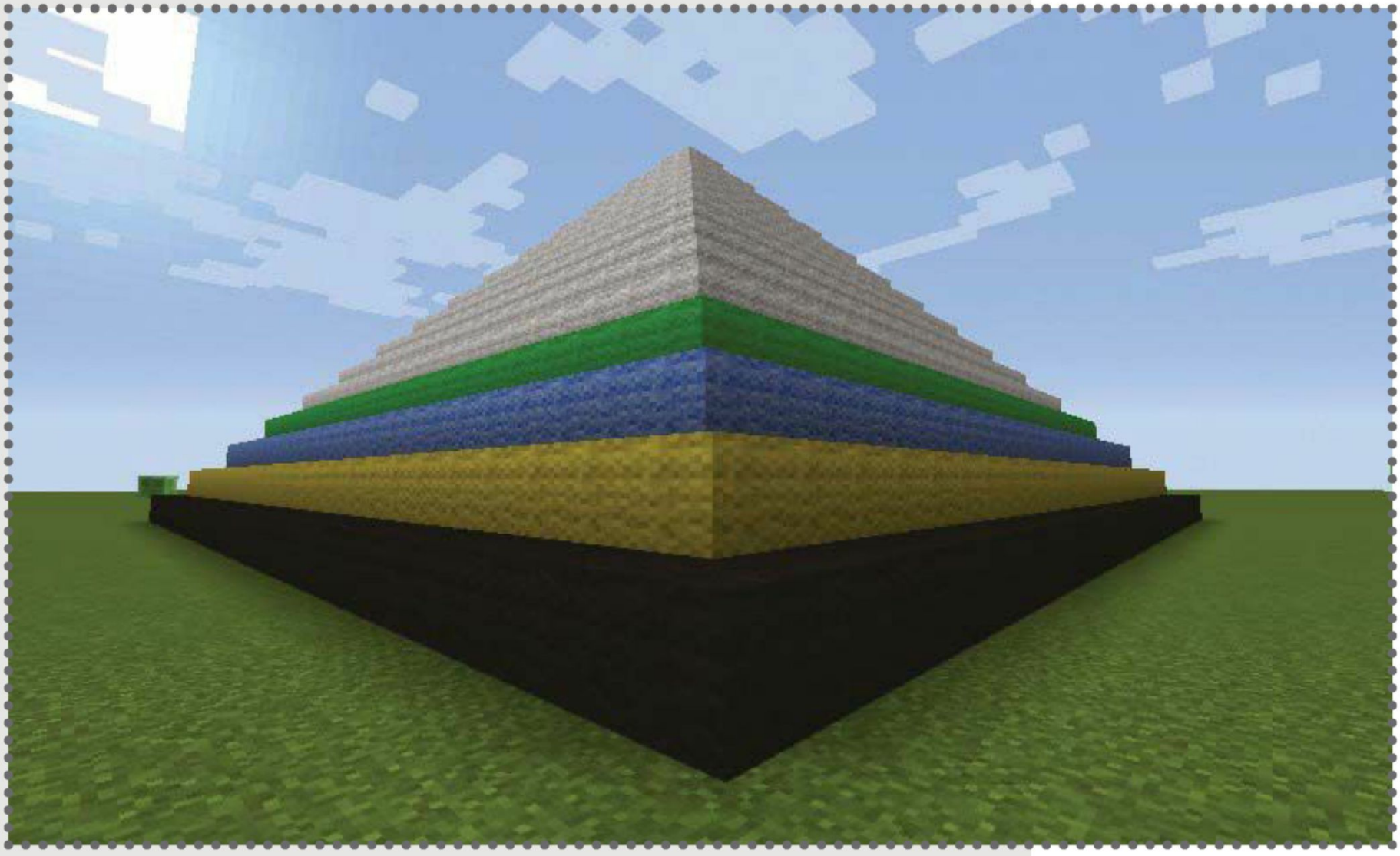
We've already initialised variables for height and count. By changing the height, we can make our pyramid larger or smaller. The count variable is used in the loop, to make sure our pyramid stops on the number of rows we specified in height. We start at 0 and increment upwards with each cycle of the loop.

If we wrap our setBlock command in conditional (if … else) statements, we can alternate rows, using the count variable to see if the row is an even number or an odd. We can place a different blockID on each row.

05 Wrap with conditional statements

Replace the current mc.setBlock(blockX, blockY, blockZ, blockID, blockType) line with the following code:

```
  if count % 2 == 0:
    mc.setBlock(blockX, blockY, blockZ,
woolBlockBlack, woolBlockBlackType)
  else:
    mc.setBlock(blockX, blockY, blockZ,
woolBlockGreen, woolBlockGreenType)
```



It just so happens we're using different coloured wool blocks. You might want to name your variables differently.

06 Alternating colours on our pyramid

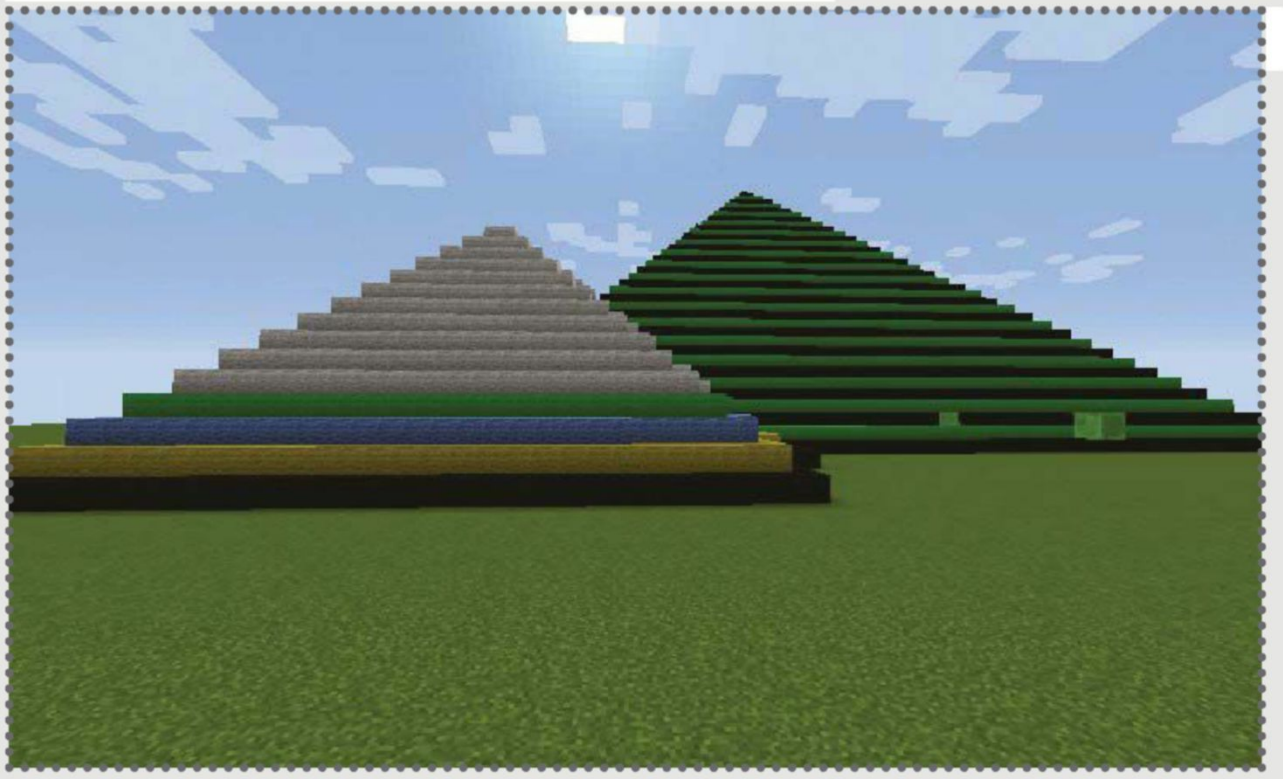
Rather than setting the colour based on odds/evens, we can get more creative and make each row a different colour. Alter the if statement to include elifs for each row that you'd like to colour, but don't forget to initialise blockID/blockType variables:

```
  woolBlock = 35
  woolBlockWhiteType = 0
  woolBlockGreenType = 5
```

07 Coding multicoloured lines

Now let's build layers upon layers.

```
if count == 0:
    mc.setBlock(blockX,      blockY,      blockZ,
woolBlock, woolBlockBlackType)
  elif count == 1:
    mc.setBlock(blockX,      blockY,      blockZ,
woolBlock, woolBlockYellowType)
```



```
elif count == 2:
    mc.setBlock(blockX,      blockY,      blockZ,
woolBlock, woolBlockBlueType)
  elif count == 3:
    mc.setBlock(blockX,      blockY,      blockZ,
woolBlock, woolBlockGreenType)
  else:
    mc.setBlock(blockX,      blockY,      blockZ,
woolBlock, woolBlockWhiteType)
```

08 Variables for the above code

Having a fancy if statement is great, but don't forget the variables:

```
  woolBlock = 35
  woolBlockGreenType = 5
  woolBlockBlackType = 15
  woolBlockYellowType = 4
  woolBlockBlueType = 3
  woolBlockWhiteType = 0
```

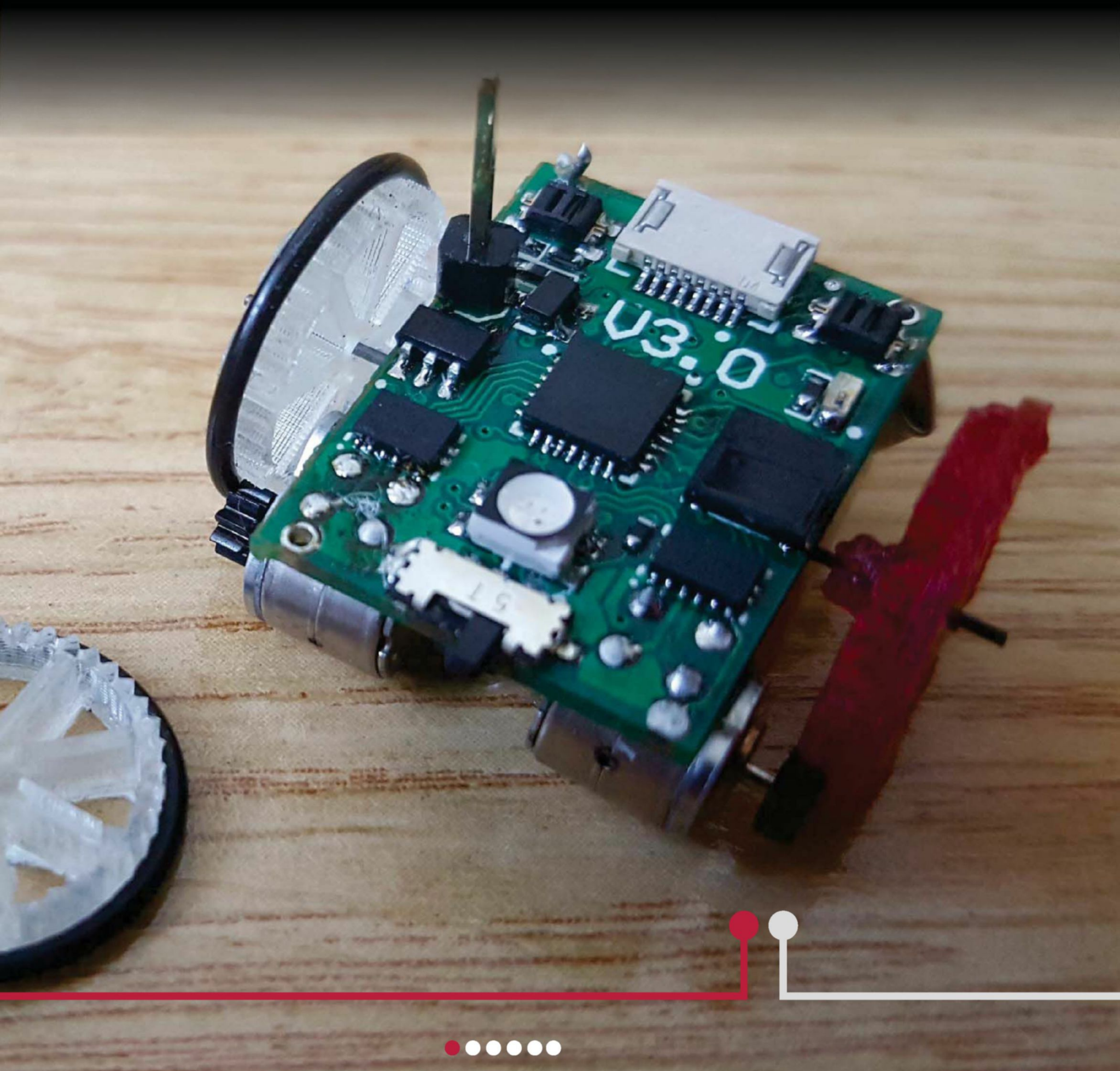
09 Egyptian influence

We should now be in a position to alter the height of our pyramids and the types of block used for each row, or to alternate the colour of rows depending on what suits our needs. Now is the time to get really creative and put that together to produce something original.

You could also try reversing the for loops, to create an inverted pyramid.

Micro robots

A pragmatic approach to teaching robotics has led to a project to build robots smaller than pocket change



Can you give us an overview of the micro robots projects? What's the idea behind your micro robots?

The micro robots project was formed when discussing how the Imperial Robotics Society could develop a course for teaching higher-level multi-robot behaviour. We have a very successful course introducing the basics of robotics on a small robot platform, roughly the size of an A5 sheet of paper, but robots of this size quickly become a problem if you want to control a load of them at once. The area you have to cordon off becomes prohibitive; also, generally you can only have one set that the class must share.

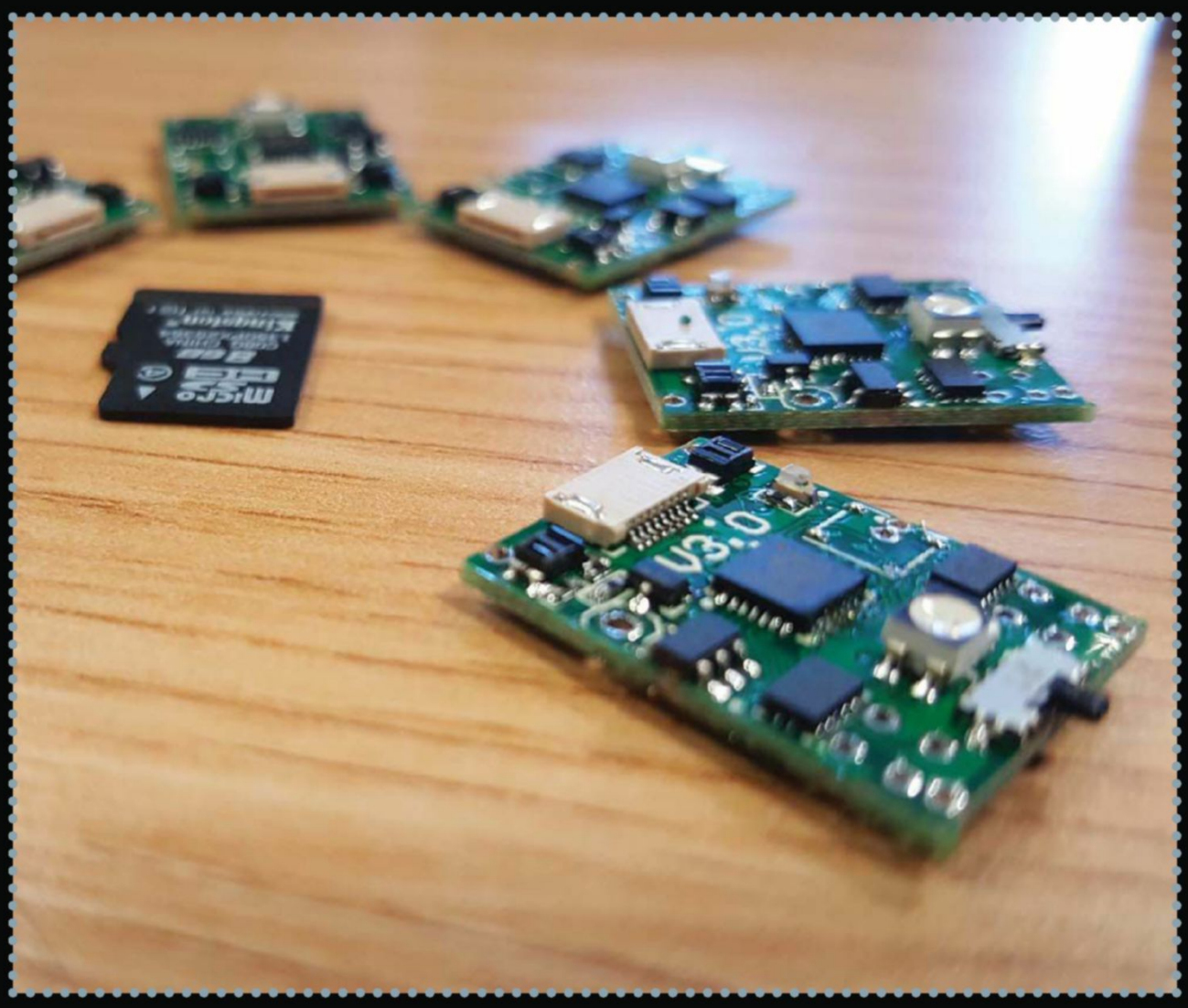
We decided that this course would not need access to the low-level hardware, as that would have been covered in the previous course, so we can use the full power of miniaturisation to reduce cost and size. We hope that in using very small robots to teach robotic behaviour classes, we can have multiple systems available for testing and have to use less space for the arenas. Additionally, the low cost of highly integrated electronics that could be assembled automatically could lower the burden on volunteer instructors. Naturally, this seed for the project has given rise to a multi-year development effort for me and my hobby partner Dr Thomas Branch.

You've recently mentioned using a camera, QR and OpenCV for tracking the robots – can you explain how this works?

For most robotic experiments, knowing the location of individual robots is a fundamental piece of

Joshua Elsdon is currently a PhD candidate at Imperial College London, studying in the field of handheld robotics and augmented reality. He's been a keen tinkerer all his life, starting with audio engineering and high-voltage circuits as a teenager which has developed into a passion for robotics.

information. The issue is that the sensors on board the robots are not sophisticated enough to discover their location from the environment. So a typical solution is to have an overhead camera keep track of where the robots are to provide input to the navigation algorithms. With a fixed camera this can be achieved reasonably simply as the system's coordinates can be based relative to the camera and the size the robots will appear in the image can be hard-coded. Though due to the fun I have had whipping the robots out at opportune moments, I wanted the system to be possible to deploy completely ad hoc. Therefore, we have implemented a system that uses a QR code like marker in the scene as a coordinate reference that provides the computer vision system with a sense of scale and orientation. The camera does not need to be orthogonal to the surface, and we can even use the location of the camera as input to the system.



THE PROJECT ESSENTIALS

Robot

STM32F031

2× forward-facing IR proximity sensors and 1× downward-facing IR line sensor

2× micro stepper motors

IR uplink and downlink modulated at 38kHz

System

ST-Nucleo based IR bridge for communication between master and robots

Master Linux system (RPi or laptop)

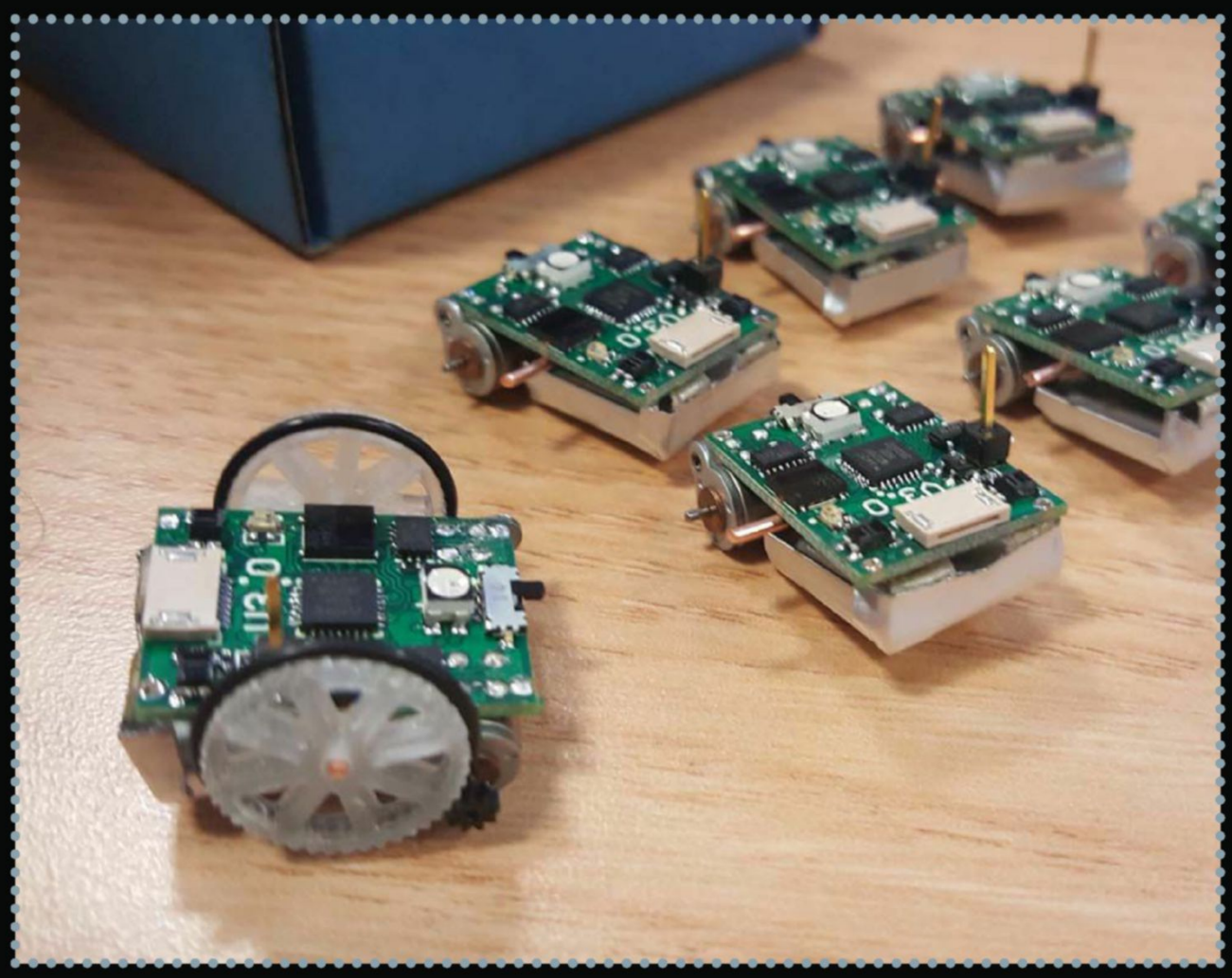
User input such as joystick

You also mention using the Raspberry Pi 3; how does that fit into project?

Originally we were thinking of this as a business case for providing educational kits, which are very price sensitive. Using the Raspberry Pi jumped out as a method of supplying the complete computational system with no setup for the user. We were aiming for the cost price of a robotic system with six robots and master computer to be roughly £100. Though because we are still doing lots of development on the project, we primarily use a full desktop system, for convenience.

Have any other interesting projects come out of the micro robots project and the training you've been running at Imperial?

Currently the robots are not used in teaching at Imperial, though in the future we hope to change

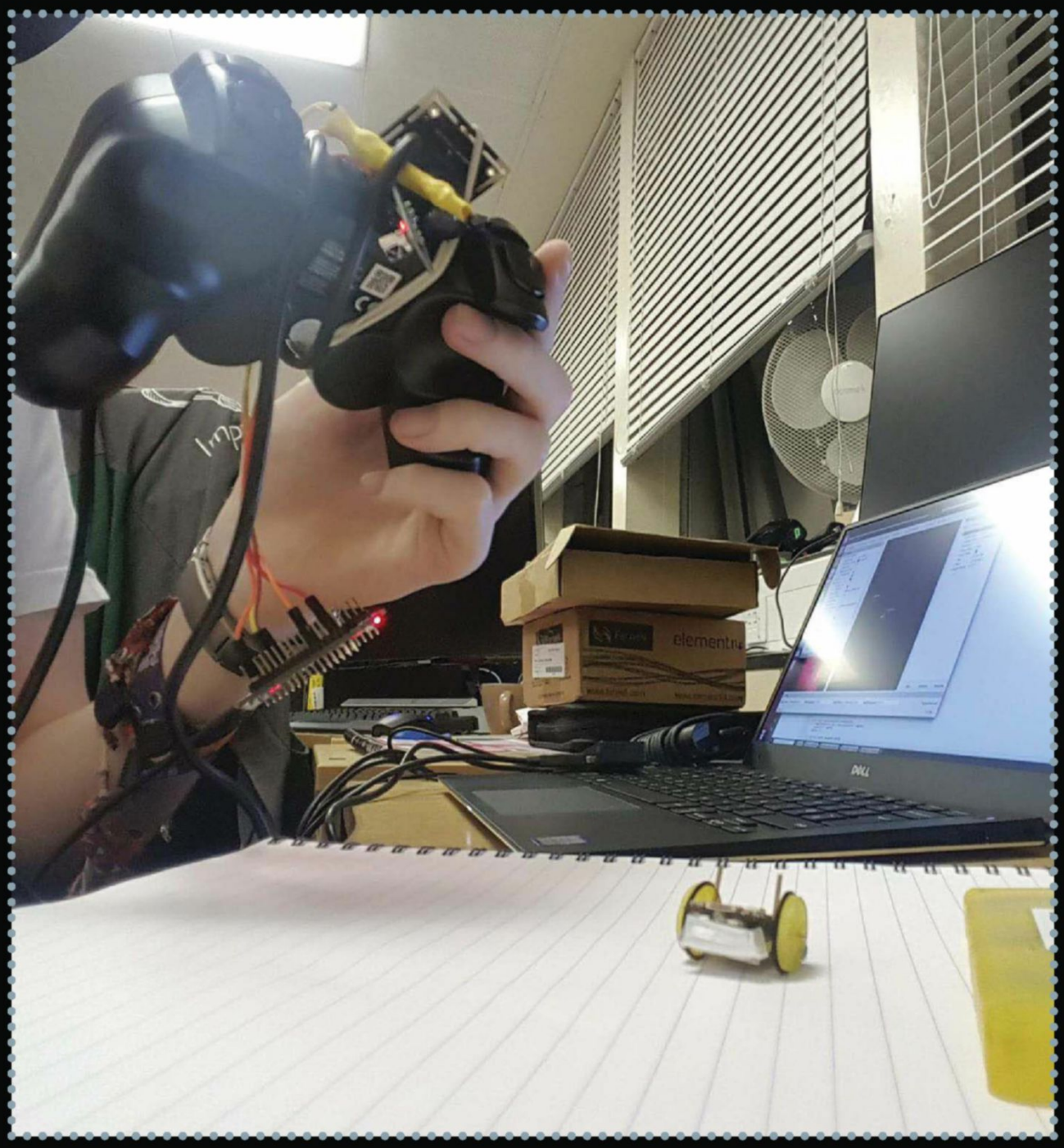


Left Version 6.0 of the micro robot kit is almost ready to go (rendered, above) and includes an accelerator for running the robots on walls: "We have demonstrated the fact that the robots can drive on a ferromagnetic surface mounted to a wall," Joshua tells us. "The accelerometer will provide all robots with a reliable absolute orientation relative to gravity."

that. I am using them in my private tutoring sessions with two 13-year-old boys. We use the robots for fun programming exercises, and we use a larger version of the robots for teaching SMD soldering techniques. The primary guiding project is to implement robotic football, though I always try and let the students have input on where the project goes, so we will have to wait and see what we actually implement.

Can you tell us about the robot HAT you're working on?

We had a couple of spare pins on the programming



Left A recent trip to a large robotics conference saw the micro robots well received, but did highlight a few problems: "The lesson learned [...] was that any demo of the project should be portable and quick to set up," says Joshua. For future trips he intends to integrate his calibrated system with controlled lighting and a mount for the camera into a single-board computer

connector for the robot, so we decided to break out an I2C interface for expansion boards. As a proof of concept, we are partially through implementing a TOF (time of flight) laser scanner on the front of the robot. Due to the precise nature of the stepper motors we use for the drive motors, we should be able to collect a 360-degree laser scan of the environment from a robot. This can be used for SLAM (simultaneous location and mapping) which, if we can pull it off, would be by far the smallest robots to complete this task.

You mention that v6 is ready for manufacture? Is there a kit coming out soon?

Yes V6.0 is more or less ready to go; it implements an accelerometer for our new idea, running the robots on walls. We have demonstrated the fact that the robots can drive on a ferromagnetic surface mounted to a wall; the accelerometer will provide all robots with a reliable absolute orientation relative to gravity. As far as kits, it seems unlikely that there would be a kit any time soon – everything you need to know is open source; only the batteries are a pain to get. We are likely to make a larger batch this year for a demonstration of the system, and perhaps that would lead to some opportunity where the robots can be supplied publicly.

Like it?

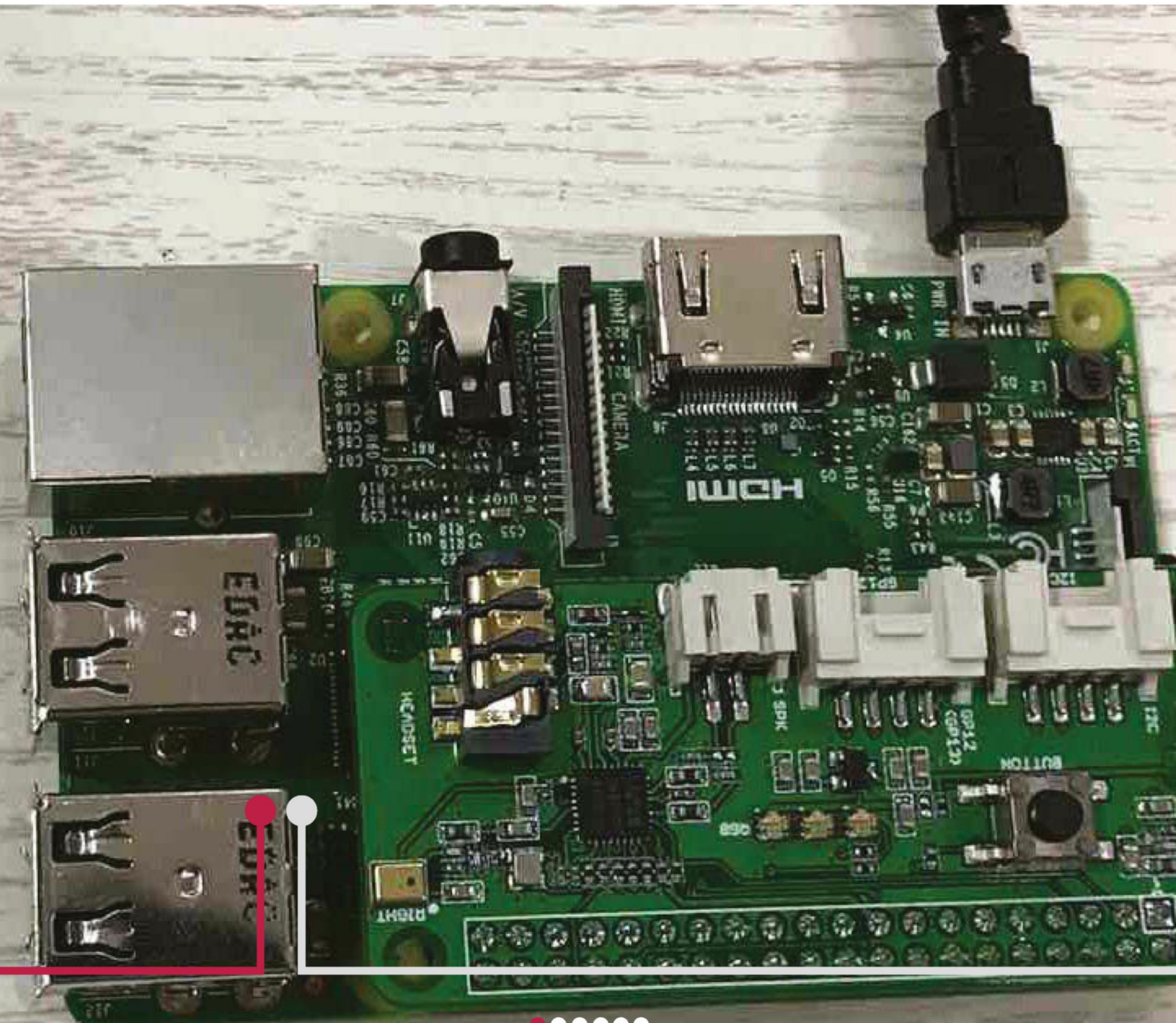
Head to Joshua's YouTube channel for demos of his camera-based location system and very tiny robots following lines, performing synchronised patterns and generally whizzing about: **http://bit.ly/JEmicrorobots**

Further reading

To keep up to date with Joshua Elsdon's micro robots project and the release of version 6 of the kit for under £100, head to **http://bit.ly/HdayMicroRobots**

Raspberry Pi AI

Use Google Assistant with your Raspberry Pi to endow it with voice-activated artificial intelligence



Google Assistant is a virtual personal assistant, designed to make your life easier by scheduling meetings, running searches and even displaying data. In this guide you'll learn how to integrate Google Assistant with your Pi and run a demo which can respond to your voice commands. As the Pi doesn't have a built in microphone, we've used the ReSpeaker 2-Mics Pi HAT, which is designed specifically for AI and voice applications. The ReSpeaker HAT costs just £12 and is compatible with the Pi Zero as well as the Pi 2B and 3B. However, if you have a USB microphone, you can use this instead. By default, Google Assistant only responds to commands when prompted with a hotword – "Hey Google!". This tutorial assumes that you have a clean install of the most recent version of Raspbian (Stretch). Once the install is complete, make sure to open Terminal and run sudo apt-get update then sudo apt-get upgrade to bring your system up to date.

THE PROJECT ESSENTIALS

Resources

- **ReSpeaker Pi drivers**
  http://bit.ly/GHReSpeaker
- **ReSpeaker 2-Mics pHAT**
  http://bit.ly/ReSpeakerpHAT
- **ReSpeaker documentation**
  http://bit.ly/ReSpeakerWiki

01 Connect the ReSpeaker

Mount the ReSpeaker 2-Mics Pi HAT on your Raspberry Pi, making sure the GPIO pins are properly aligned. If connected properly, the red LED will illuminate. Connect your headphones or speakers to the 3.5mm audio jack on the device. If your speakers aren't powered, connect another micro-USB cable to the power port on the ReSpeaker to supply enough current. The microphones themselves are located at either end of the HAT, labelled 'Left' and 'Right'.

02 Install device drivers

Return to the Terminal on your Pi and run git clone --depth=1 https://github.com/respeaker/seeed-

voicecard. Switch to this directory with the command cd seeed-voicecard, then launch the installer with sudo ./install.sh 2mic. Once installation is complete, reboot the Pi by running sudo reboot. Reopen a Terminal and run the command aplay -l to list all hardware playback devices. You should see the ReSpeaker listed under card 1 as 'seeed2micvoicec'. Next, run the command arecord -l to list all sound capture devices to check that 'seeed2micvoicec' is listed here too.

03 Configure sound device

Right-click the volume icon at the top right of your screen and make sure that the voicecard is selected. Right-click once again and choose 'USB Device Settings'. In the new window which opens, click the drop-down menu marked 'Sound Card' and choose the seeed voicecard or another preferred audio device. Next, click 'Make Default' to ensure the Pi will keep using this device next time it restarts. Don't worry for now about the lack of volume controls; simply click 'OK' to save and exit.

04 Adjust sound levels

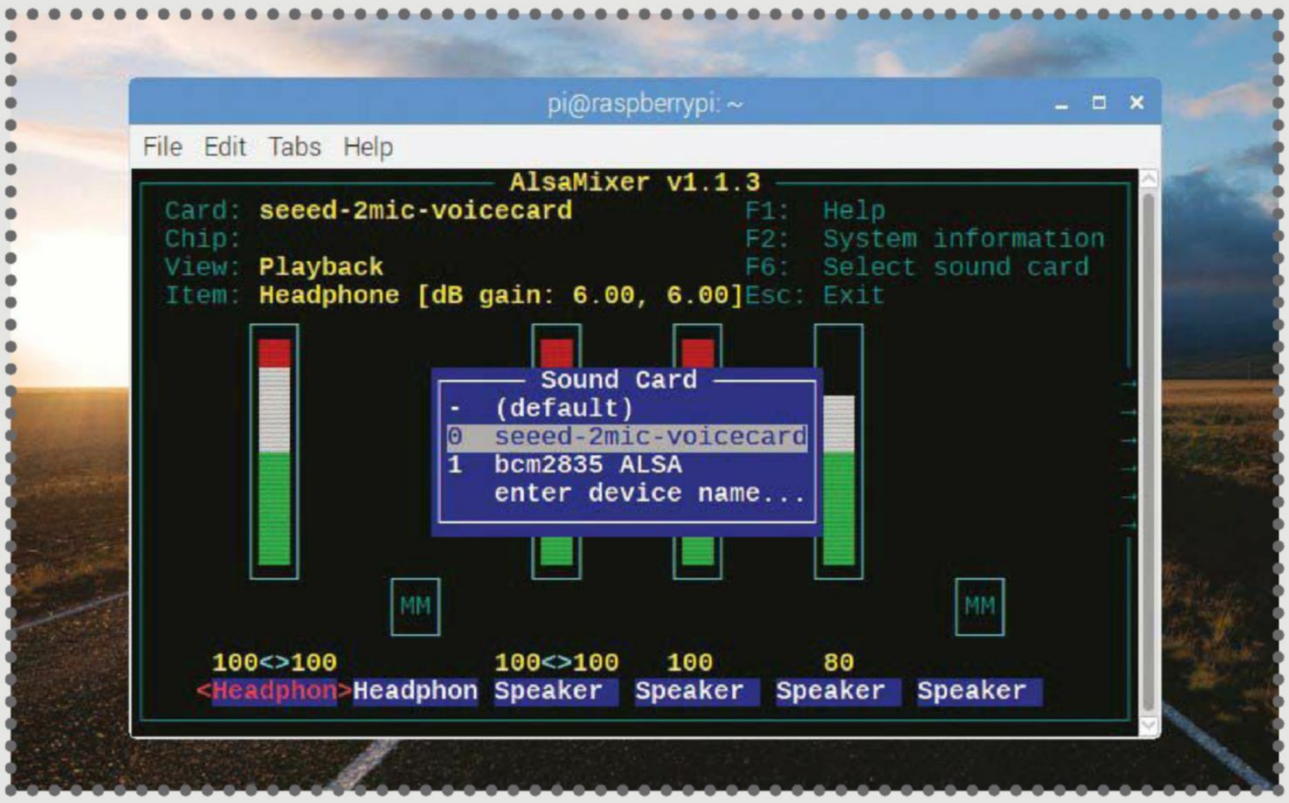
Return to the Terminal and run the command sudo alsamixer. This handy utility lets you adjust the volume for playback and recording of all hardware devices. First press F6 and use the arrow to select your chosen audio device, which should be listed under Card 1. Use Return to select it, then press F5 to list all volume settings. Use the left/right arrow keys to select different controls and up/down to adjust individually. Press Esc to quit, then run sudo alsactl store to save the settings.

05 Visit Google Cloud Platform

In order to interface your device with Google Assistant, you'll need to have a Google account. If you don't have one, head over to https://gmail.com in your web browser of choice and click 'Create an Account' at the top right. Once you've done so, visit https://console.cloud.google.com/project and sign in with the account details. Click 'Yes', then 'Accept'. Next, visit http://bit.ly/GoogleCP. Click 'Enable'. A page will appear entitled 'To view this page, select a project.' Click 'Create' and name your project. Make a note of the project ID, then click 'Create' again.

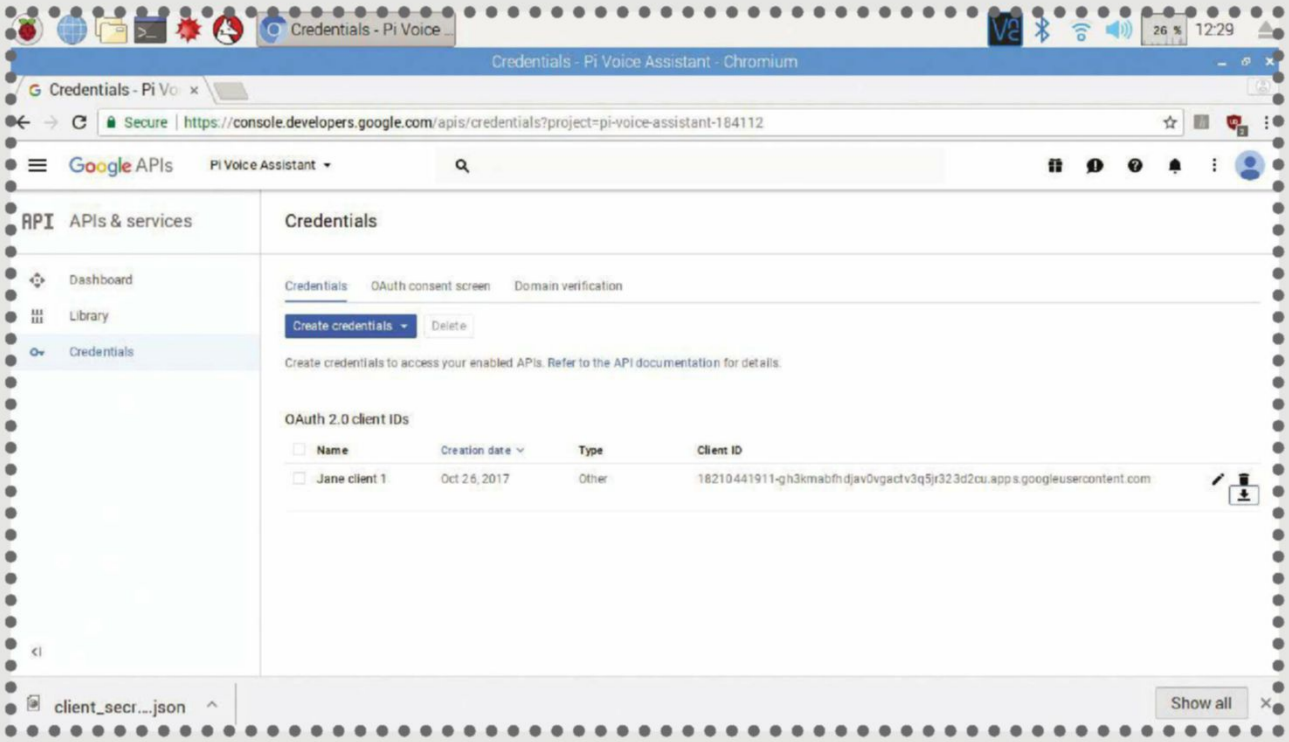
06 Create product

In order to use the Google Assistant API, you need to generate credentials. These are stored in your Pi and let Google know you're using an authorised device. Click 'Enable' on the API page which has now loaded. Next, visit http://bit.ly/GCP-OAuth. Click 'Configure Consent' at the top right and enter a product name such as 'Jane's Pi Voice Assistant'. If you're creating a marketable application, fill in the homepage, logo and privacy policy fields. Click 'Save' at the bottom when you're done.



07 Create client ID

Google will return you to the 'Create Client ID' screen. Under 'Application Type', choose 'Other', then click 'Create'. You will now see your Client ID and secret key. Ignore these for now and click 'OK'. Google will now return you to the 'Credentials' screen where your client ID will now be listed. Click the down arrow on the right-hand side to save the credentials file with the extension .json. Once the file is downloaded, open your file browser and move it to /home/pi. This will make it easier to access.



08 Authorise Google Assistant

In your web browser, visit this address: https://myaccount.google.com/activitycontrols. Make sure to switch on 'Web & App Activity', 'Device Information' and 'Voice & Audio Activity'.

Next, open Terminal and run sudo pip install --upgrade google-auth-oauthlib[tool]. You can use this tool to authorise Google Assistant to work with your Pi, using the JSON file you downloaded in the previous step. To do this, run the command:

google-oauthlib-tool --client-secrets /home/pi/client_secret_.json --scope https://www.googleapis.com/auth/assistant-sdk-prototype --save -headless.

Make sure to replace client_secret_.json with the actual name of the JSON file.

09 Enter confirmation code

The Google OAuth tool will now ask you to visit a URL to obtain a confirmation code. Right-click and copy the link. Open your web browser and right-click the address bar and choose 'Paste and Go'. Click on your own account name and then choose to 'Allow' the voice assistant. The page will generate a code. Copy this from your browser and paste it into the Terminal prompt. If successful, you'll see a message stating that your credentials have been saved.

10 Start Google Assistant demo

In Terminal, run the command sudo apt-get install pulseaudio, then launch it with pulseaudio &. Finally, run the command google-assistant-demo. Make sure that your headphones or speakers are connected. Use the default hotwords "Hey Google" to have the Assistant listen, then ask it a standard question such as "What's the weather like in London today?" to hear a mini weather forecast.

Google Assistant can also play sounds; e.g. "What does a dog sound like?" It also performs conversions such as asking telling you the value of 100 euros in dollars.

Make an egg-drop game with the Sense HAT

Use the same hardware that Major Tim Peake used on the ISS and code your own drop-and-catch game

Some of the most basic and repetitive games are the most fun to play. Consider Flappy Bird, noughts and crosses or even catch. This tutorial shows you how to create a simple drop-and-catch game that makes excellent use of some of the Sense HAT's features. Start off by coding an egg – a yellow LED – to drop each second, and a basket – a brown LED – on the bottom row of LEDs. Use the Sense HAT's accelerometer to read and relay back when you tilt your Sense HAT left or right, enabling you move the basket toward the egg. Successfully catch the egg and you play again, with a new egg being dropped from a random position… But, if you miss one, then it breaks and it's game over! Your program will keep you up to date with how you are progressing, and when the game ends, your final score is displayed. If you don't own a Sense HAT, you can use the emulator that is available on the Raspbian with PIXEL operating system. You can also see the Egg Drop game in action here: **youtube.com/watch?v=QmjHMzuWIqI**

01 Import the modules

First, open your Python editor and import the SenseHAT module, line 1. Then import the time module, line 2, so you can add pauses to the program. The random module, line 3, is used to select a random location from the top of the LEDs, from which the egg will drop. To save time typing 'SenseHAT' repeatedly, add it to a variable, line 4. Finally, set all the LEDs to off to remove the previous score and game data, line 5.

```
from sense_hat import SenseHat
import time
import random
sense = SenseHat()
sense.clear()
```

02 Set the variables

Next, create the variables to hold the various game data. On line 1, create a global variable to hold the status of the game. This records whether the game is in play or has ended. The global enables the status to be used later on in the game with other parts of the program. On line 2, create another variable to hold your game score. Set the game_over variable on line 3 to False, this means the game is not over. The position of each LED on the matrix is referred to by the co-ordinates x and y, with the top line being number 0 down to number 7 at the bottom. Create a variable to hold the position of the basket, which is set on the bottom line of the LEDs, number 7. Finally, set the score to zero.

```
global game_over
```

Using a pretty web domain

If this is the first time you have set up a web server, you are probably now thinking about how you can access the website from outside of your home network and with a domain name of your choice. Unfortunately, we do not have space to cover that here, but you can easily do this with a dynamic DNS service and a custom domain of your choosing. Dig around on Google and you will find plenty of advice.

```
  global score
  game_over = False
  basket_x = 7
  score =  0
```

03 Measure the basket movement: part 1

The basket is controlled by tilting your Sense HAT to the left or right, which alters the pitch. Create a function to hold the code, which will be used to respond to the movement and move the basket. On line 1, name the function; include the pitch reading and the position of the basket, basket_x. Use sense.set_pixel to turn on one LED at the bottom-right of the LEDs matrix, the co-ordinates (7,7), line 2. Then set the next position of the basket to the current position so that the function is updated when it runs again. This updates the variable with the new position of the basket and turns on the corresponding LED. This has the effect of looking like the basket has moved.

04 Measure the basket movement: part 2

The second part of the function consists of a conditional which checks the pitch and the basket's current position. If the pitch is between a value of 1-179 and the basket is not at position zero, then the Sense HAT is tilted to the right and therefore the basket is moving to the right. The second condition checks that the value is between 359 and 179, which means that the tilt is to the left, line 3. The last line of code returns the x position of the basket so it can be used later in the code – see Step 13.

```
  if 1 < pitch < 179 and basket_x != 0:
      new_x -= 1
  elif 359 > pitch > 179 and basket_x != 7:
```

```
        new_x += 1
    return new_x,
```

05 Create images for your game

Images are built up of pixels that combine to create an overall picture. Each LED on the matrix can be automatically set from an image file. For example, an image of a chicken can be loaded, the colours and positions calculated, and then the corresponding LEDs enabled. The image needs to be 8×8 pixels in size so that it fits the LED matrix. Download the test picture file, chicken.png, and save it into the same folder as your program. Use the code here in a new Python window to open and load the image of the chicken (line 3). The Sense HAT will do the rest of the hard work for you.

```
from sense_hat import SenseHat
sense = SenseHat()
sense.load_image("chicken.png")
```

06 Create your own 8×8 image

The simplest method to create your own image with the

LEDs is a superb on-screen program that enables you to manipulate the LEDs in real-time. You can change the colours, rotate them and then export the image as code or as an 8×8 PNG file.

First, you need to install Python PNG library; open the Terminal window and type:

```
sudo pip3 install pypng
```

After this has finished, type:

```
git clone https://github.com/jrobinson-uk/
RPi_8x8GridDraw
```

Once the installation has completed, move to the RPi folder:

```
cd RPi_8x8GridDraw
```

Now enter the command:

```
python3 sense_grid.py
```

…to run the application.

07 Create and export your image

The Grid Editor enables you to select from a range of colours displayed down the right-hand side of the window. Simply choose the colour and then click the location of the LED on the grid; select 'Play on LEDs' to display the colour on the Sense HAT LED. Clear the LEDs using the Clear Grid button and then start over. Finally, when exporting the image, you can either save as a PNG file and then apply the code in the previous step to display the picture, or you can export the layout as code and import that into your program.

08 Display a message: the game begins

Now you have an image, you are ready to create the function that controls the whole game. Create a new function, line 1, called main, and add the code:

```
sense.show_message
```

…to display a welcome message to the game, line 3. The values 255, 255 and 0 refer to the colour of the message (in this example, yellow). Edit these to choose your own preferred colour.

```
def main():
    global game_over
    sense.show_message("Egg Drop", text_
colour = [255, 255, 0])
```

09 Display your start image

Once the start message has scrolled across the Sense HAT LED matrix, you can display your game image, in this example a chicken. Due to the orientation of the Sense HAT and the location of the wires, you'll need to rotate it through 90 degrees so it faces the player, line 1. Load the image with the code sense.load.image, line 2. Display the image for a few seconds using time.sleep(), line 3. Note that the lines from now on are indented in line with the previous line.

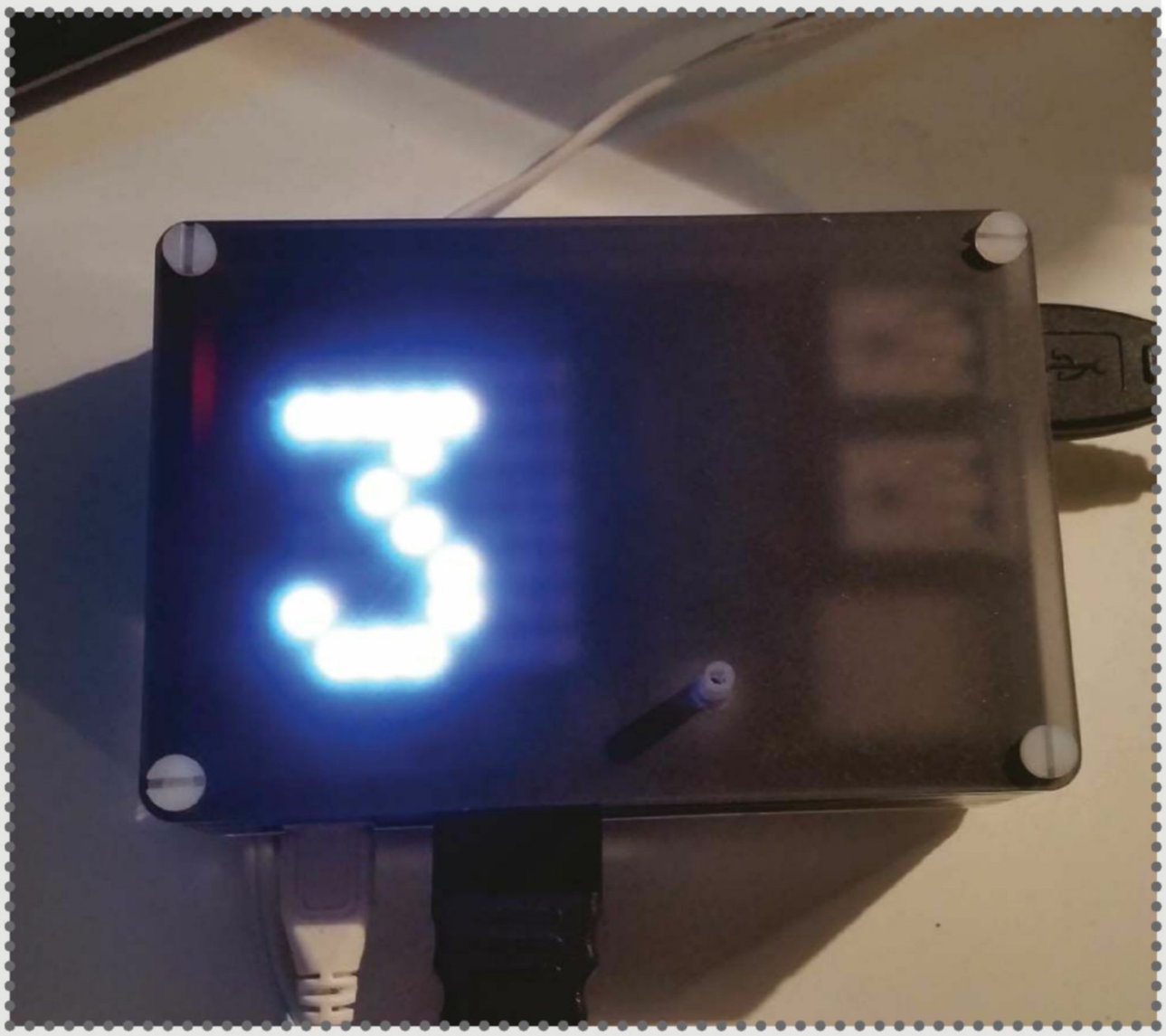
```
    sense.set_rotation(90)
    sense.load_image("chick.png")
    time.sleep(2)
```

```
        sense.set_rotation()
```

10 Count down to the game starting

Once the start image has been displayed, prepare the player to get ready for the game with a simple countdown from three to one. First create a list called countdown, which stores the values 3, 2, and 1, line 1. Use a for loop, line 2, to iterate through each of the numbers and display them. This uses the code sense.show_message(str(i) to display each number on the LEDs. You can adjust the colour of the number using the three-part RGB values, text_colour = [255, 255, 255]), line 3.

```
  countdown = [3, 2, 1]
```



```
    for i in countdown:
        sense.show_message(str(i), text_colour =
    [255, 255, 255])
```

11 Set the egg and basket

As the game starts, set the horizontal position, the 'x' position, of the basket to 7; this places the basket in the bottom right-hand corner of the LED matrix. Now set the x position of the egg at a random positon between 0 and 7, line 2. This is at the top of the LED matrix and ensures that the egg does not always fall from the same starting point. Last, set the egg's y value to 0 to ensure that the egg falls from the very top of the LED matrix, line 3.

```
  basket_x = 7
  egg_x = random.randrange(0,7)
  egg_y = 0
```

12 Display the egg and basket

In the previous step, you set the positions for the egg and the basket. Now use these variables to display them. On line 1, set the egg using the code sense.set.pixel followed by its x and y co-ordinates. The x position is a random position between 0 and 7, and the y is set to 0 to ensure that the egg starts from the top. Next, set the colour to yellow. (Unless your egg is rotten, in which case set it to green (0,255, 0). Next, set the basket position using the same code, line 2, where the x position is set to 7 to ensure that the basket is displayed in the bottom right-hand LED. Set the colour to brown using the values 139, 69, 19.

```
    sense.set_pixel(egg_x, egg_y, [255, 255, 0])
    sense.set_pixel(basket_x, 7, [139, 69, 19])
    time.sleep(1)
```

13 Move the basket: part 1

Begin by checking that the game is still in play (the egg is still dropping), checking that the game_over variable is False, line 1. On line 2, import the score. Next, take a reading of the 'pitch' of the Sense HAT, using the code sense.get_orientation()['pitch'], line 3. Note that this is the value derived from the function you created in steps 4 and 5. The final line of code uses the function to turn off the LED that represents the basket and then looks at the value of the pitch, determining if the Sense HAT is tilted to the left or right, and then either adds or subtracts one from the x position of the current LED. This has the effect of selecting either the adjacent left or right LED to the current LED. Finally, update the basket_x value with the new position value.

14 Move the basket: part 2

Your program has now calculated the new position of the basket. Next, turn on the relevant LED and display the basket in its new position. On line 1, use the code sense.set_pixel(basket_x, 7, [139, 69, 19]) to set and turn on the LED; basket_x is the value calculated in the previous step using the function in steps 4 and 5. Add a short time delay to avoid over-reading the pitch, line 2. You now have a basket that you can move left and right.

```
sense.set_pixel(basket_x, 7, [139, 69, 19])
time.sleep(0.2)
```

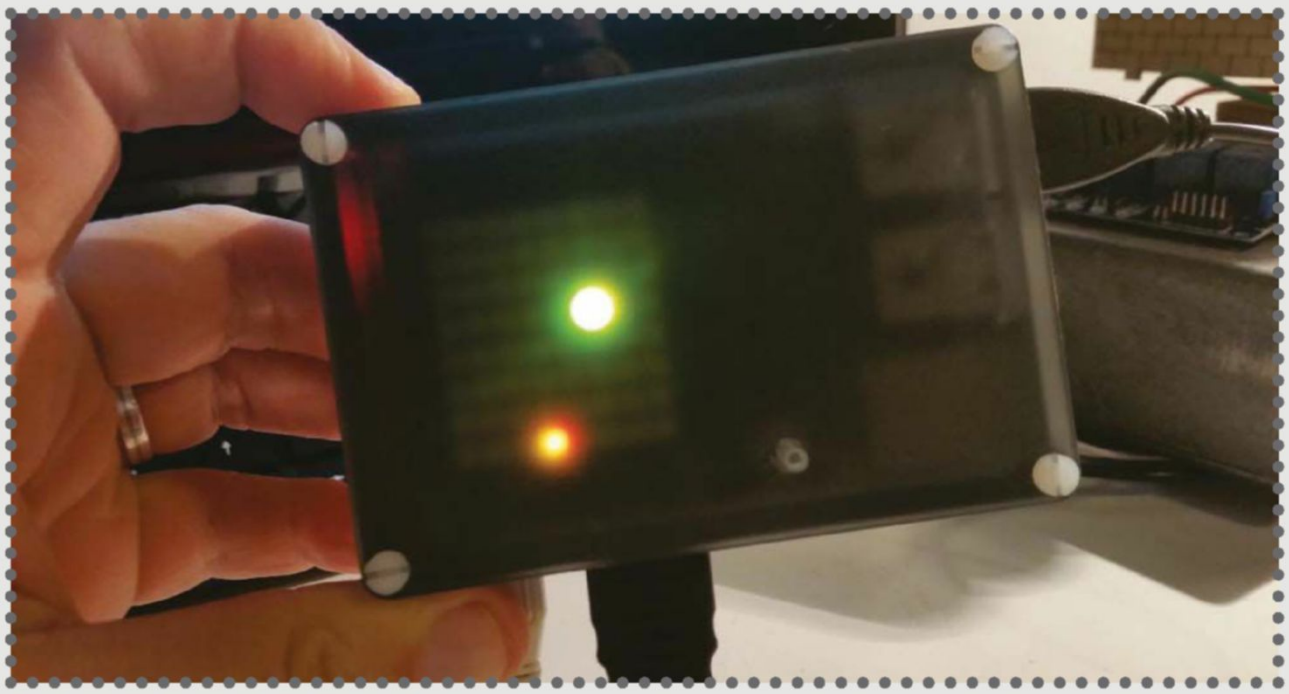
15 Drop the egg: part 1

The egg is dropped from a random position from one of the LEDs across the top line. To make it appear to be dropping, first turn off the LED that represents the egg using the code sense.set_pixel(egg_x, egg_y, [0, 0, 0]). The values 0, 0, 0, refer to black and therefore no colour will be displayed; it will appear that the egg is no longer on the top line.

```
sense.set_pixel(egg_x, egg_y, [0, 0, 0])
```

16 Drop the egg: part 2

The Grid Editor enables you to select from a range of colours displayed down the right-hand side of the window. Simply choose the colour and then click the location of the LED on the grid; select 'Play on LEDs' to display the colour on the Sense HAT LED. Clear the LEDs using the Clear Grid button and then start over. Finally, when exporting the image, you can either save as a PNG file and then apply the code in the previous step to display the picture, or you can export the layout as code and import that into your program.



17 Did you catch the egg?

At this stage in the tutorial, you have a falling egg and a basket that you can move left and right as you tilt the Sense HAT. The purpose of the game is to catch the egg in the basket, so create a line of code to check that this has happened, line 1. This checks that the egg is at the bottom of the LED matrix, ie in position 7, and that the basket's x position is the same value as the egg's x position. This means that the egg and the basket are both located in the same place and therefore you have caught the egg.

```
if (egg_y == 7) and (basket_x == egg_x or basket_x-1 ==
    egg_x ):
```

18 Success!

If you catch the egg in the basket, then you gain one point. Notify the player by scrolling a message across the LEDs using the sense.show.message(), line 1. Write your own message and select a colour. Since you have caught the egg, it should disappear as it is in the basket; to do this set the 'egg' pixel to a colour value of 0, 0, 0. This basically turns off the egg LED, making the egg disappear. Note that these lines are both indented.

```
    sense.show_message("1up", text_colour = [0, 255, 0])
    sense.set_pixel(egg_x, egg_y, [0, 0, 0])
```

19 Set up for the next round

Since you caught the egg, you get to play the game again. Set a new egg to drop from a random x position on the LED matrix, line 1. Update your score by one point and then set the egg's y position to 0, line 3. This ensures that the egg is back at the very top of the LED matrix

before it starts dropping.

```
  egg_x = random.randrange(0,7)
  score = score =+ 1
  egg_y = 0
```

20 What happens if you miss the egg?

If you miss the egg or drop it, then the game ends. Create a conditional to check that the egg's y position is equal to or greater than 7, line 1. Display a message that states that the game is over, line 2, and then return the value of the score that is displayed across the LED matrix in step 23.

```
  elif egg_y == 7:
      sense.show_message("Game Over", text_
    colour = [255, 38, 0])
      return score
```

21 Stop the game

Since the game is over, change the game_over variable to True, which stops the game loop from running again and then runs the last line of the program.

```
      game_over = True
      break
```

22 Start the game

The main instructions and game mechanics are stored in one function called main(), which holds most of the game structure and processes. Functions are located at the start of a program to ensure that they are loaded first, ready for the program to use. To start the game, simply call the function (line 1), add a small delay (line 2), and ensure all the LEDs are set to off before the game

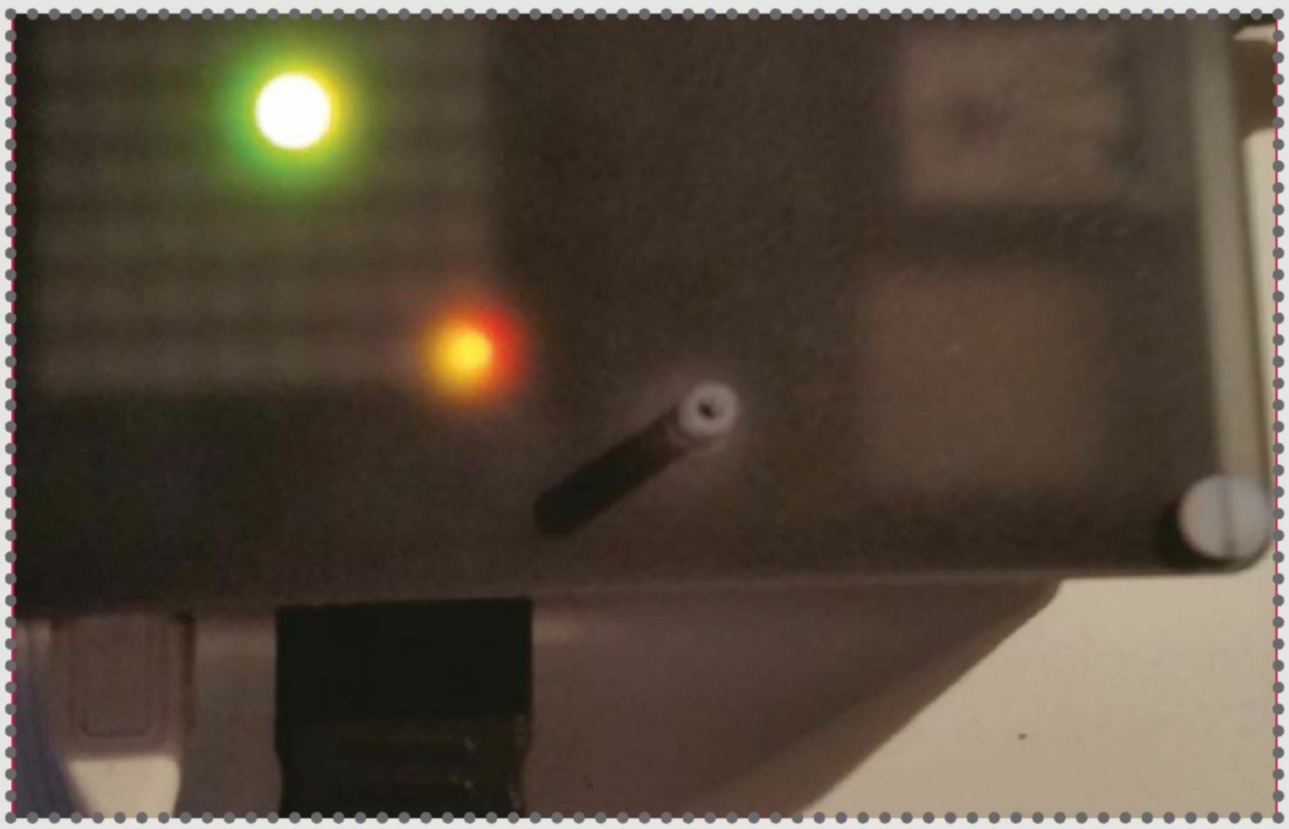
starts (line 3).

```
main()
time.sleep(1)
sense.clear()
```

23 Display your final score

If you did not catch the egg, then the game is over and your score is scrolled across the LEDs. This uses the line sense.show_message and then pulls the value from the global_score variable; convert this value into a string using str, line 1. Your program is now completed; save the file and then run it. Press F5 on the keyboard to do this. After the opening image and message are displayed, the countdown will begin and the game will start. Can you catch the egg?

```
sense.show_message("You Scored " +
  str(score),
  text_colour = [128, 45, 255], scroll_speed
  = 0.08)
```



Full code listing

```
from sense_hat import SenseHat
###Egg Drop###
###Coded by dan_aldred###

import time
import random
sense = SenseHat()
sense.clear()

global game_over
global score

game_over = False
basket_x = 7
score =  0

'''main pitch measurement'''
def basket_move(pitch, basket_x):
    sense.set_pixel(basket_x, 7, [0, 0, 0])
    new_x = basket_x
    if 1 < pitch < 179 and basket_x != 0:
        new_x -= 1
    elif 359 > pitch > 179 and basket_x != 7:
        new_x += 1
    return new_x,

'''Main game setup'''
def main():
    global game_over

    '''Introduction'''
    sense.show_message("Egg Drop", text_colour
    = [255, 255, 0])
```

```
    sense.set_rotation(90)
    sense.load_image("chick.png")
    time.sleep(2)
    sense.set_rotation()

    '''countdown'''
    countdown = [3, 2, 1]
    for i in countdown:
        sense.show_message(str(i), text_colour
= [255, 255, 255])

    basket_x = 7
    egg_x = random.randrange(0,7)
    egg_y = 0
    sense.set_pixel(egg_x, egg_y, [255, 255,
0])
    sense.set_pixel(basket_x, 7, [139, 69, 19])
    time.sleep(1)

    while game_over == False:
        global score
        '''move basket  first'''
        '''Get basket position'''
        pitch = sense.get_orientation()['pitch']
        basket_x, = basket_move(pitch,
basket_x)

        '''Set Basket Positon'''
        sense.set_pixel(basket_x, 7, [139, 69,
19])
        time.sleep(0.2)

        '''Egg drop'''
```

```
        sense.set_pixel(basket_x, 7, [0, 0, 0])
        sense.set_pixel(egg_x, egg_y, [0, 0,
    0])
        egg_y = egg_y + 1
        #print (egg_y)
        sense.set_pixel(egg_x, egg_y, [255,
    255, 0])

        '''Check posiion of the egg and basket
    x , y'''
        if (egg_y == 7) and (basket_x ==
    egg_x or basket_x-1 == egg_x ):
            sense.show_message("1up", text_
    colour = [0, 255, 0])
            sense.set_pixel(egg_x, egg_y, [0,
    0, 0])#hides old egg
            egg_x = random.randrange(0,7)
            score = score =+ 1
            egg_y = 0

        elif egg_y == 7:
            sense.show_message("Game Over",
    text_colour = [255, 38, 0])
            return score
            game_over = True
            break

main()
time.sleep(1)
sense.clear()
sense.show_message("You Scored " + str(score),
    text_colour = [128, 45, 255], scroll_speed
    = 0.08)
```

Make a Raspberry Pi-based warrant canary

Protect yourself from Orwellian gagging orders and secret warrants with your own warrant canary

The warrant canary borrows its name from the unfortunate bird that was taken down mine shafts. It's a reaction against secret requests to obtain customer's personal data.

For example, when a court order is served on a business, its owners are forbidden from alerting users that their data has been compromised. Warrant canaries cleverly slalom around this legal hurdle through regularly making a statement that they have not been subject to such a request. If they switch off their warrant canary, as Reddit did in March 2016, users will know that an external agency also has access to the data a company stores about them. The legal ramifications of this are complex and depend on where you are in the world; this tutorial is an exercise in proof-of-concept only.

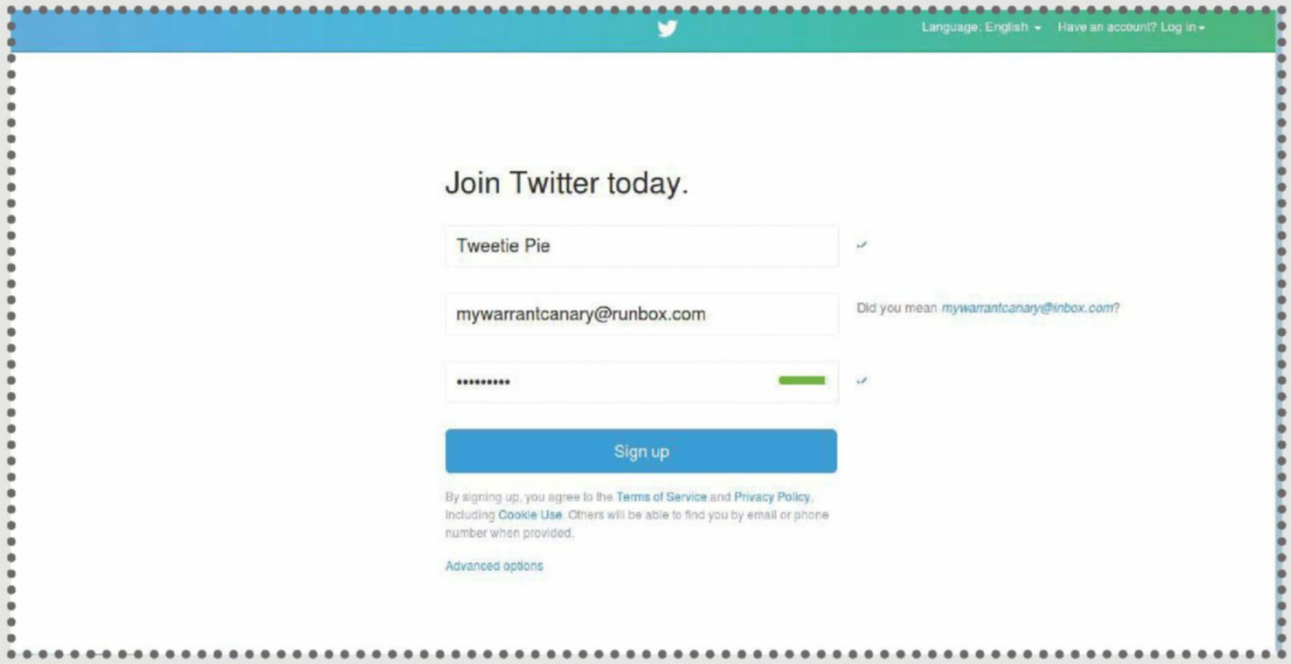
For this tutorial we will use the Raspberry Pi along with a dedicated Twitter bot to build your own warrant canary.

01 Create a Twitter account

Head to Twitter.com and create a new Twitter account. You will need a valid e-mail address and mobile phone number. If you already have a Twitter account, make sure to add a phone number, as this is required in order to use a Twitter Bot. See **https://support.twitter.com/articles/110250#** for help with this.

02 Create the application

Once your account is set up, go to http://apps.twitter.com on any advice and click Create New Application to begin setting up a dedicated bot for your warrant canary. Under 'personal website', enter the URL of your Twitter channel. Tick to say you've read and agreed to the developer agreement then click Create your Twitter



Application.

03 Create your Access Token

Click on Keys and Access Tokens. Make a note of your API keys. Next, scroll to Token Actions and click Create My Access Token. Write down the Access Token and the Access Token secret. Your Raspberry Pi will need this information in order to be able to connect securely to Twitter later.

04 Tweak profile

Optionally at this stage you can choose to delete your mobile phone number from Twitter. It is only required once in order to deter spammers. Feel free at this stage to add a profile picture to your account and update the bio to explain what your warrant canary is for.

05 Install Software on the Pi

Open Terminal on your Pi or connect via SSH. Run…

```
sudo apt-get update
```

Left Use a valid e-mail address and phone number to create a new account

Best Practices

Twitter has strict safeguards against spam bots. This is why it requires accounts using Applications to be verified by SMS. You may notice when testing your script that Twitter will also not allow duplicate statuses. Make sure that your tweets are well spaced apart. The above tutorial will have your Pi's canary tweet every day at midnight. If you need them to be closer together the raspberrypi.org website has some tips on using Twython to post messages at random from a list.

```
  and
  sudo apt-get upgrade
  to update Raspbian. Next run
  sudo pip install twython requests
requests_oauthlib
```

…to install all necessary software.

06 Create the Python script

In the Pi Terminal, run the command…

```
  sudo
```

```
…to create your script. Next paste in the
following:
```

```
  #!/usr/bin/env python
  import sys
  from twython import Twython

  tweetStr = “I have not been subject
to any government gagging orders and/or
subpoenas at the time of this Tweet.”

  # Insert your API keys and access tokens
here
  apiKey = ‘yourapikey’
  apiSecret = ‘yourapisecret’
  accessToken = ‘youraccesstoken’
  accessTokenSecret =
‘youraccesstokensecret’

  api = Twython(apiKey,apiSecret,accessToken
```

```
,accessTokenSecret)

  api.update_status(status=tweetStr)

  print “Tweeted: “ + tweetStr
```

Replace “yourapikey” and so on with the actual values from the page http://apps.twitter.com. Press Ctrl+X, then Y, then Return to save and exit.

07 Test the script

Run this command…

```
  python canary.py
```

…to test the script. If successful it will display a message saying that the Tweet has been sent.

08 Schedule your canary

The warrant canary should be set to tweet daily unless you intervene. In the Pi terminal run the following…

```
  sudo crontab -e
```

If this is the first time you’ve run crontab, choose option 2 to select an editor.
Scroll to the bottom and paste the following:

```
 0 0 * * * /usr/bin/python /home/pi/canary.
py
```

09 Add a photo to your Tweets (Optional)

Run

```
sudo nano canary.py
```

…to edit your Python script. Comment out the line beginning “tweetStr”, then replace the lines…

```
  “api.update_status(status=tweetStr)

  print “Tweeted: “ + tweetStr@
```

…with:

```
  message = “No FBI here!”
  with open(‘/home/pi/Downloads/image.jpg’,
‘rb’) as photo:
      api.update_status_with_
media(status=message, media=photo)

  print “Tweeted: “ + message
```

```
 GNU nano 2.2.6                File: canary.py

#!/usr/bin/env python
import sys
from twython import Twython

#tweetStr = "I have not been subject to any government gagging orders and/or su

# Insert your API keys and access tokens here
apiKey = 'toYFAO4cQkEWnWXmBNVlzEM6H'
apiSecret = 'BO53G8O5GNqIMChE3LCIx37MDcYnCgfFewXCofNVS8dJFXlAST'
accessToken = '807645065707331584-iyAzK5VlI5pdei6SCsqRKQ8AcDzjySj'
accessTokenSecret = 'fUsyFRqGguGyXlo1eWWu9xynDpt5iDLBO3PwGJu6gNOHC'

api = Twython(apiKey,apiSecret,accessToken,accessTokenSecret)

message = "No FBI here!"
with open('/home/pi/Pictures/canary.jpg', 'rb') as photo:
    api.update_status_with_media(status=message, media=photo)

print "Tweeted: " + message
```

Are warrant canaries legal?

The legal loophole that warrant canaries supposedly exploit has yet to be tested. In 2015 Australia outlawed them altogether and other countries may follow suit. Prosecution would be difficult however without revealing the existence of the very information the original warrant was designed to suppress in open court. It’s fine to make one as a proof of concept, but you’ll need to research to find out if actually using one as an injunction warning system is legal where you are.

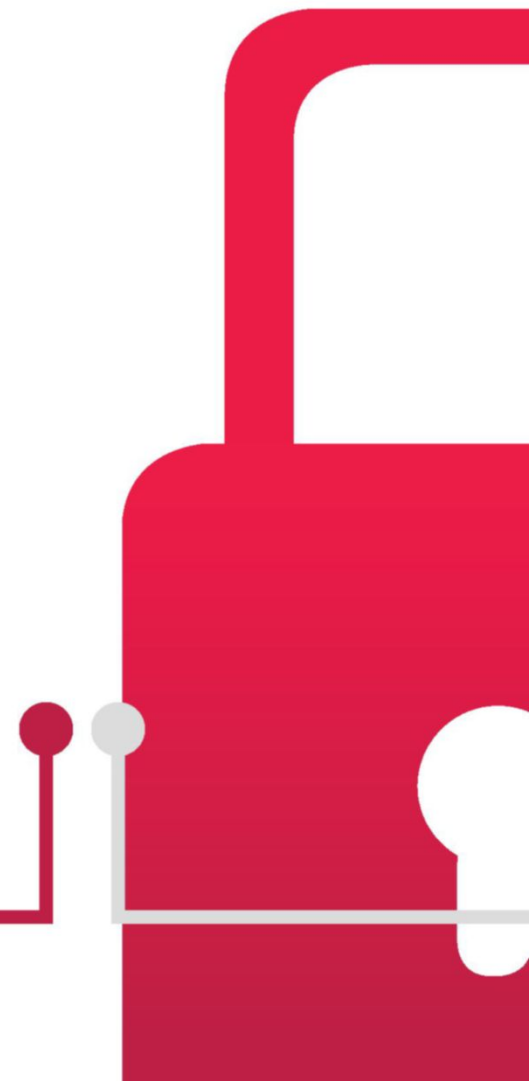
Handle multiple tasks

Your Raspberry Pi project may need to deal with more than one thing at a time. Learn how to handle multiple tasks in Python

Several articles in this column have covered different techniques to accomplish specific tasks. What we haven't covered, so far, is how best to deal with the case when your Raspberry Pi project needs to manage several different tasks concurrently. This month, we will look at how to use the multitasking capabilities within Python to manage multiple tasks. In the standard library, there are three main modules available. They are threading, multiprocessing and concurrent. Each has its own strengths and weaknesses. Since these are all part of the standard library, there should not be anything extra that you will need to install.

First, we will look at the threading module. There are two ways that you can use this module. The first is to use it to create new thread objects that can be told to run some target function within your program. The following is a simple example:

```
import threading
def my_func():
```



```
        print("Hello World")
    my_thread = threading.Thread(target=my_
func)
    my_thread.start()
```

Assuming that your tasks can be partitioned into separate functions, you can create a thread for each of these functions. One thing to be aware of is that these new threads will not start executing the function code until you call the start method. At that point, the target function will start running asynchronously in the background. You can check to see whether or not a given thread is done by using code like that below:

```
    if my_thread.is_alive():
        print('This thread is still running')
```

At some point in the main body of your program, you are going to want to use the results from the functions running in these threads. When this happens you can use the join() method of the thread object. This halts the main core of your program and forces it to wait until the thread exits. The thread exits by default when the running function exits.

But, how do you write code that uses threads well? The first item to consider is whether you will be using data that is globally available or whether you are using data that should only be visible within the current thread. If you do need local only data, you can create a local object that can store these values. The following code stores a string with your author's name in it:

```
  mydata = threading.local()
  mydata.myname = 'Joey Bernard'
```

This would be code used within the function being run by a thread. If you need to use global data, you need to consider how different threads may try to use this global data. If everyone is reading from a given variable, you won't run into any issues. The problem arises when you have multiple threads that may try to write a given variable. In this case you'll end up with a situation known as a race condition, where one thread may overwrite the data from another. In these cases, you will need to use lock objects to manage access to these global variables. A basic example would look like:

```
  mylock = threading.Lock()
  counter = 0
  def func1():
      mylock.acquire()
      counter = counter + 1
      mylock.release()
```

As you can see, you create the lock object in the main body of your program. Then, within the function code, you try to acquire the lock. If it is free, you get access to it and it is locked. If the lock object has already been locked by another thread, then this call to acquire blocks and waits until the lock has been released. This is why you need to be really careful to always have a release statement for every acquire statement. Otherwise, you'll have a bug that will be almost

Why Python?

It's the official language of the Raspberry Pi. Read the docs at python.org/doc

impossible to find after the fact. This also introduces a bottleneck to your program, so you want to make sure that whatever code exists between the acquire and lock is the bare minimum required to do the necessary work. This is the simplest form of locking mechanism available in Python. If your needs are greater, you can look at some of the other options to see if they might offer better control access.

Along with controlling access to global data, you may need to communicate directly between threads. This can be handled through an event object, which can be used to set a flag to true or false and make that visible to other threads. As an example, the code below shows how to set and use such a flag:

```
  event1 = threading.Event()
  def func1():
        ....
        event1.set()
        ....
  def func2():
        ....
        if event1.set():
            print('I got a flag from func1')
        ....
```

Sometimes, the only communication you need is to know when all of the threads have completed some stage of their work. Say, you multiple threads loading data files and you need to wait until everyone is done before moving on to the next stage. In this case, you can do so with barrier objects. Below, you can

see how you could add a barrier to the two threads (above):

```
barrier1 = threading.Barrier(2)
def func1():
    ....
    barrier1.wait()
    ....
def func2():
    ....
    barrier1.wait()
    ....
```

In the above code, you need to set how many threads will take part in the barrier object when you create it. Then, when threads use it and call the wait method, they will block until all of the threads call the wait method.

"How do you write code that uses threads well?"

The threading module is a light, fast and easy method to add the ability divide up the processing within your code, but it does suffer from one major issue. Within the Python core engine, there is a structure called the GIL (global interpreter lock). The GIL is used to control access to certain core functions and data within the Python interpreter. This means that at certain points, your threads will run only one at a time. This can introduce a serious bottleneck in some situations. If you are in this boat, then you may need to use the multiprocessing module. This module uses subprocesses to bypass the GIL completely in order to get true parallel operation. In its most basic use case, you could use something like the code below to get behaviour similar to what you get with threads:

```
  import multiprocessing
  def f(name):
      print('hello', name)

  p = multiprocessing.Process(target=f,
args=('bob',))
  p.start()
  p.join()
```

This appears to be the same on the surface, but what is happening in the back-end is radically different. The process object starts up a new Python engine in one of a number of ways. The default on UNIX systems, like the Pi, is to fork a new process. The fork method essentially makes a complete copy of the current Python engine and executes the given function. Another method is to spawn a new Python engine. In the spawn method, only the parts of the current Python engine that is needed for the new Python engine. If you do need to change it, you can use the following code:

```
  multiprocessing.set_start_method('spawn')
```

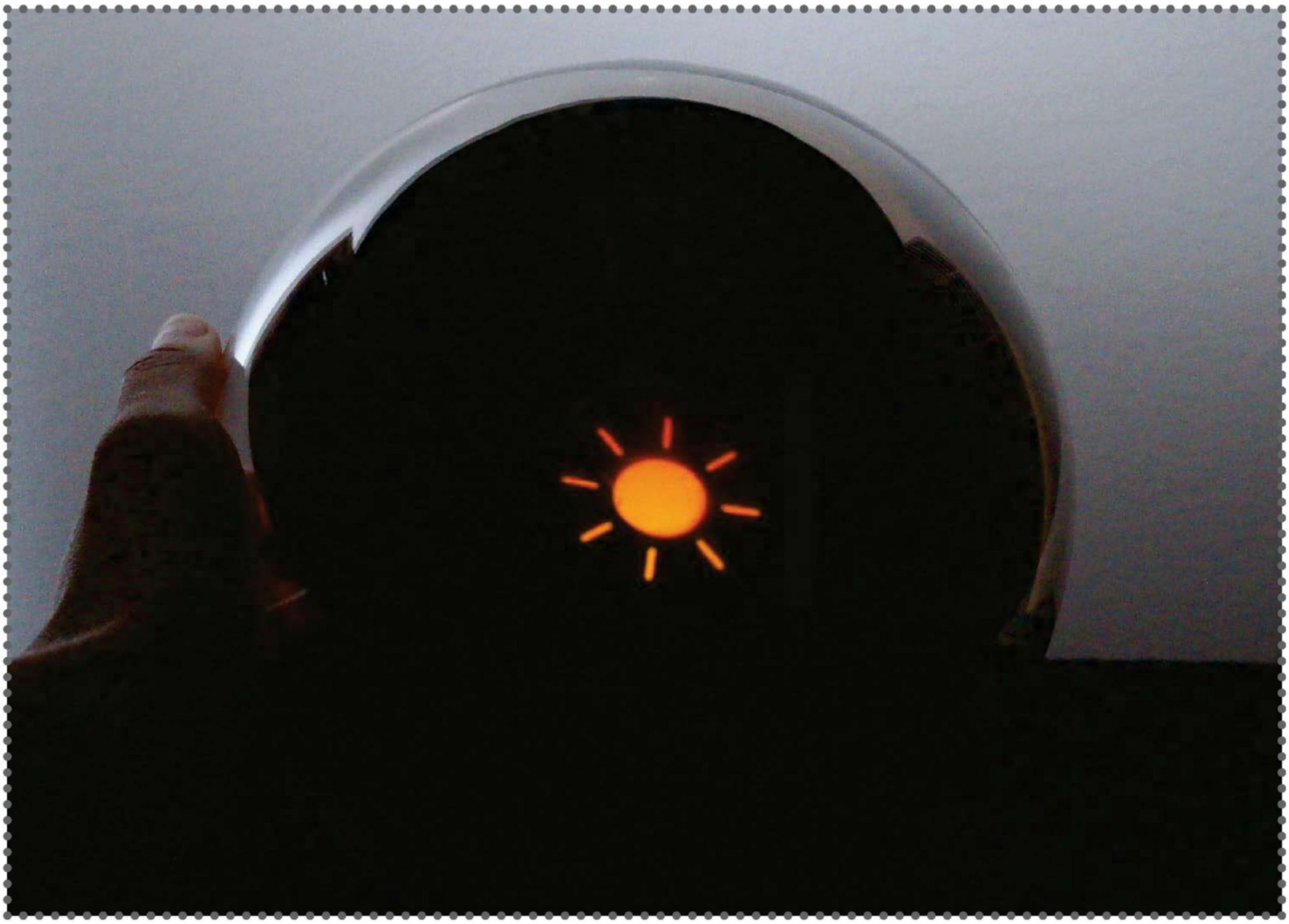
If you need to start many subprocesses, this may help speed your code up. The set_start_method should only ever be called once in a given program.

Hopefully, this article has given you some ideas on how to include the ability to manage multiple tasks in parallel. This can be a powerful tool to make the software design of your project more flexible and capable. Be aware that we have only been able to

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

"Predict the weather with your Pi"



Get this issue's source code at:

www.linuxuser.co.uk/raspicode

9012

9000